WES 237A: Introduction to Embedded System Design (Winter 2024)

Lab 3: Serial and CPU

Due: 2/4/2024 11:59pm

Ricardo Lizarraga, rlizarraga0@gmail.com *619.252.4157*	**PID: A69028483** https://github.com/RiLizarraga

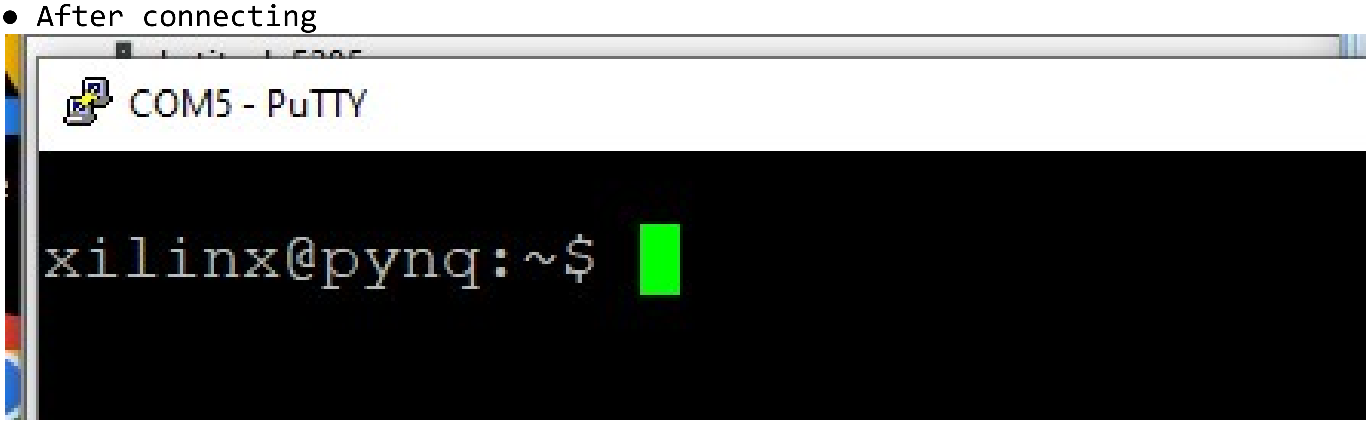
In order to report and reflect on your WES 237A labs, please complete this Post-Lab report by the end of the weekend by submitting the following 2 parts:

- Upload your lab 3 report composed by a single PDF that includes your in-lab answers to the bolded questions in the Google Doc Lab and your Jupyter Notebook code. You could either scan your written copy, or simply type your answer in this Google Doc. **However, please make sure your responses are readable.**
- Answer two short essay-like questions on your Lab experience.

All responses should be submitted to Canvas. Please also be sure to push your code to your git repo as well.

Serial Connection

- Using a micro USB cable, connect your board to your laptop
- Connect to board using the serial connection
  - Linux
    - Open a new terminal
    - Run the command
    - *sudo screen /dev/<port> 115200* #port: ttyUSB0 or ttyUSB1
  - MAC
    - Open a new terminal
    - Run the command and check the PYNQ resources for the port
    - *sudo screen /dev/<port> 115200* #port: check resources
  - Windows
    - Check the resource for how to connect through serial to the PYNQ board
  - Resources:
    - https://pynq.readthedocs.io/en/v2.0/getting_started.html
    - https://www.nengo.ai/nengo-pynq/connect.html
- After connecting



Boot loader terminal commands

```
Zynq> help
?          - alias for 'help'
base       - print or set address offset
bdinfo     - print Board Info structure
blkcache   - block cache diagnostics and control
boot       - boot default, i.e., run 'bootcmd'
bootd      - boot default, i.e., run 'bootcmd'
bootefi    - Boots an EFI payload from memory
bootelf    - Boot from an ELF image in memory
bootm      - boot application image from memory
bootp      - boot image via network using BOOTP/TFTP protocol
bootvx     - Boot vxWorks from an ELF image
bootz      - boot Linux zImage image from memory
chpart     - change active partition of a MTD device
clk        - CLK sub-system
cmp        - memory compare
coninfo    - print console devices and information
cp         - memory copy
crc32      - checksum calculation
dcache     - enable or disable data cache
dfu        - Device Firmware Upgrade
dhcp       - boot image via network using DHCP/TFTP protocol
dm         - Driver model low level access
echo       - echo args to console
editenv    - edit environment variable
efidebug   - Configure UEFI environment
env        - environment handling commands
erase      - erase FLASH memory
exit       - exit script
ext2load   - load binary file from a Ext2 filesystem
ext2ls     - list files in a directory (default /)
ext4load   - load binary file from a Ext4 filesystem
ext4ls     - list files in a directory (default /)
ext4size   - determine a file's size
ext4write  - create a file in the root directory
false      - do nothing, unsuccessfully
fatinfo    - print information about filesystem
fatload    - load binary file from a dos filesystem
fatls      - list files in a directory (default /)
fatmkdir   - create a directory
fatrm      - delete a file
fatsize    - determine a file's size
fatwrite   - write file into a dos filesystem
fdt        - flattened device tree utility commands
flinfo     - print FLASH memory information
fpga       - loadable FPGA image support
fru        - FRU table info
fstype     - Look up a filesystem type
fstypes    - List supported filesystem types
go         - start application at address 'addr'
gpio       - query and control gpio pins
help       - print command description/usage
i2c        - I2C sub-system
```

```
icache     - enable or disable instruction cache
iminfo     - print header information for application image
imls       - list all images found in flash
imxtract   - extract a part of a multi-image
itest      - return true/false on integer compare
led        - manage LEDs
ln         - Create a symbolic link
load       - load binary file from a filesystem
loadb      - load binary file over serial line (kermit mode)
loads      - load S-Record file over serial line
loadx      - load binary file over serial line (xmodem mode)
loady      - load binary file over serial line (ymodem mode)
loop       - infinite loop on address range
ls         - list files in a directory (default /)
md         - memory display
mdio       - MDIO utility commands
mii        - MII utility commands
mm         - memory modify (auto-incrementing address)
mmc        - MMC sub system
mmcinfo    - display MMC info
mtd        - MTD utils
mtdparts   - define flash/nand partitions
mtest      - simple RAM read/write test
mw         - memory write (fill)
nand       - NAND sub-system
nboot      - boot from NAND device
net        - NET sub-system
nfs        - boot image via network using NFS protocol
nm         - memory modify (constant address)
panic      - Panic with optional message
part       - disk partition related commands
ping       - send ICMP ECHO_REQUEST to network host
printenv   - print environment variables
protect    - enable or disable FLASH write protection
pxe        - commands to get and boot from pxe files
random     - fill memory with random pattern
reset      - Perform RESET of the CPU
run        - run commands in an environment variable
save       - save file to a filesystem
saveenv    - save environment variables to persistent storage
setenv     - set environment variables
sf         - SPI flash sub-system
showvar    - print local hushshell variables
size       - determine a file's size
sleep      - delay execution for some time
source     - run script from memory
spl        - SPL configuration
sysboot    - command to get and boot from syslinux files
test       - minimal test like /bin/sh
tftpboot   - boot image via network using TFTP protocol
tftpput    - TFTP put command, for uploading files to a server
thordown   - TIZEN "THOR" downloader
time       - run commands and summarize execution time
timer      - access the system timer
true       - do nothing, successfully
ubi        - ubi commands
ubifsload  - load file from an UBIFS filesystem
ubifsls    - list files in a directory
ubifsmount- mount UBIFS volume
ubifsumount- unmount UBIFS volume
usb        - USB sub-system
usbboot    - boot from USB device
version    - print monitor, compiler and linker version
zynq       - Zynq specific commands
```

- Restart the board (*$ sudo reboot*)
- Interrupt the boot (keyboard interrupt)
- List current settings (*printenv*)
- **Put a screenshot of your *$ printenv* output**

```
Zynq> printenv
arch=arm
baudrate=115200
board=zynq
board_name=zynq
boot_a_script=load ${devtype} ${devnum}:${distro_bootpart} ${scriptaddr} ${prefix}${script}; source ${scriptaddr}
boot_efi_binary=load ${devtype} ${devnum}:${distro_bootpart} ${kernel_addr_r} efi/boot/bootarm.efi; if fdt addr ${fdt_addr_r}; th
en bootefi ${kernel_addr_r} ${fdt_addr_r};else bootefi ${kernel_addr_r} ${fdtcontroladdr};fi
boot_efi_bootmgr=if fdt addr ${fdt_addr_r}; then bootefi bootmgr ${fdt_addr_r};else bootefi bootmgr;fi
boot_extlinux=sysboot ${devtype} ${devnum}:${distro_bootpart} any ${scriptaddr} ${prefix}${boot_syslinux_conf}
boot_net_usb_start=usb start
boot_prefixes=/ /boot/
boot_script_dhcp=boot.scr.uimg
boot_scripts=boot.scr.uimg boot.scr
boot_syslinux_conf=extlinux/extlinux.conf
boot_targets=mmc0 jtag mmc0 mmc1 qspi nand nor usb0 usb1 pxe dhcp
bootcmd=run distro_bootcmd
bootcmd_dhcp=devtype=dhcp; run boot_net_usb_start; if dhcp ${scriptaddr} ${boot_script_dhcp}; then source ${scriptaddr}; fi;seten
v efi_fdtfile ${fdtfile}; if test -z "${fdtfile}" -a -n "${soc}"; then setenv efi_fdtfile ${soc}-${board}${boardver}.dtb; fi; set
env efi_old_vci ${bootp_vci};setenv efi_old_arch ${bootp_arch};setenv bootp_vci PXEClient:Arch:00010:UNDI:003000;setenv bootp_arc
h 0xa;if dhcp ${kernel_addr_r}; then tftpboot ${fdt_addr_r} dtb/${efi_fdtfile};if fdt addr ${fdt_addr_r}; then bootefi ${kernel_a
ddr_r} ${fdt_addr_r}; else bootefi ${kernel_addr_r} ${fdtcontroladdr};fi;fi;setenv bootp_vci ${efi_old_vci};setenv bootp_arch ${e
fi_old_arch};setenv efi_fdtfile;setenv efi_old_arch;setenv efi_old_vci;
bootcmd_jtag=echo JTAG: Trying to boot script at ${scriptaddr} && source ${scriptaddr}; echo JTAG: SCRIPT FAILED: continuing...;
bootcmd_mmc0=devnum=0; run mmc_boot
bootcmd_mmc1=devnum=1; run mmc_boot
bootcmd_nand=nand info && nand read ${scriptaddr} ${script_offset_f} ${script_size_f} && echo NAND: Trying to boot script at ${sc
riptaddr} && source ${scriptaddr}; echo NAND: SCRIPT FAILED: continuing...;
bootcmd_nor=cp.b ${script_offset_nor} ${scriptaddr} ${script_size_f} && echo NOR: Trying to boot script at ${scriptaddr} && sourc
e ${scriptaddr}; echo NOR: SCRIPT FAILED: continuing...;
bootcmd_pxe=run boot_net_usb_start; dhcp; if pxe get; then pxe boot; fi
bootcmd_qspi=sf probe 0 0 0 && sf read ${scriptaddr} ${script_offset_f} ${script_size_f} && echo QSPI: Trying to boot script at $
{scriptaddr} && source ${scriptaddr}; echo QSPI: SCRIPT FAILED: continuing...;
bootcmd_usb0=devnum=0; run usb_boot
bootcmd_usb1=devnum=1; run usb_boot
bootcmd_usb_dfu0=setenv dfu_alt_info boot.scr ram $scriptaddr $script_size_f && dfu 0 ram 0 60 && echo DFU0: Trying to boot scrip
t at ${scriptaddr} && source ${scriptaddr}; echo DFU0: SCRIPT FAILED: continuing...;
bootcmd_usb_dfu1=setenv dfu_alt_info boot.scr ram $scriptaddr $script_size_f && dfu 1 ram 1 60 && echo DFU1: Trying to boot scrip
t at ${scriptaddr} && source ${scriptaddr}; echo DFU1: SCRIPT FAILED: continuing...;
bootcmd_usb_thor0=setenv dfu_alt_info boot.scr ram $scriptaddr $script_size_f && thordown 0 ram 0 && echo THOR0: Trying to boot s
cript at ${scriptaddr} && source ${scriptaddr}; echo THOR0: SCRIPT FAILED: continuing...;
bootcmd_usb_thor1=setenv dfu_alt_info boot.scr ram $scriptaddr $script_size_f && thordown 1 ram 1 && echo THOR1: Trying to boot s
cript at ${scriptaddr} && source ${scriptaddr}; echo THOR1: SCRIPT FAILED: continuing...;
bootdelay=2
bootm_low=0
bootm_size=20000000
cpu=armv7
distro_bootcmd=for target in ${boot_targets}; do run bootcmd_${target}; done
efi_dtb_prefixes=/ /dtb/ /dtb/current/
ethaddr=00:00:05:6b:03:3a
fdt_addr_r=0x1f00000
fdtcontroladdr=1eae1160
kernel_addr_r=0x2000000
load_efi_dtb=load ${devtype} ${devnum}:${distro_bootpart} ${fdt_addr_r} ${prefix}${efi_fdtfile}
loadaddr=0x0
mmc_boot=if mmc dev ${devnum}; then devtype=mmc; run scan_dev_for_boot_part; fi
modeboot=sdboot
pxefile_addr_r=0x2000000
ramdisk_addr_r=0x3100000
scan_dev_for_boot=echo Scanning ${devtype} ${devnum}:${distro_bootpart}...; for prefix in ${boot_prefixes}; do run scan_dev_for_e
xtlinux; run scan_dev_for_scripts; done;run scan_dev_for_efi;
scan_dev_for_boot_part=part list ${devtype} ${devnum} -bootable devplist; env exists devplist || setenv devplist 1; for distro_bo
otpart in ${devplist}; do if fstype ${devtype} ${devnum}:${distro_bootpart} bootfstype; then run scan_dev_for_boot; fi; done; set
env devplist
scan_dev_for_efi=setenv efi_fdtfile ${fdtfile}; if test -z "${fdtfile}" -a -n "${soc}"; then setenv efi_fdtfile ${soc}-${board}${
boardver}.dtb; fi; for prefix in ${efi_dtb_prefixes}; do if test -e ${devtype} ${devnum}:${distro_bootpart} ${prefix}${efi_fdtfil
e}; then run load_efi_dtb; fi;done;run boot_efi_bootmgr;if test -e ${devtype} ${devnum}:${distro_bootpart} efi/boot/bootarm.efi;
then echo Found EFI removable media binary efi/boot/bootarm.efi; run boot_efi_binary; echo EFI LOAD FAILED: continuing...; fi; se
tenv efi_fdtfile
scan_dev_for_extlinux=if test -e ${devtype} ${devnum}:${distro_bootpart} ${prefix}${boot_syslinux_conf}; then echo Found ${prefix
}${boot_syslinux_conf}; run boot_extlinux; echo SCRIPT FAILED: continuing...; fi
scan_dev_for_scripts=for script in ${boot_scripts}; do if test -e ${devtype} ${devnum}:${distro_bootpart} ${prefix}${script}; the
n echo Found U-Boot script ${prefix}${script}; run boot_a_script; echo SCRIPT FAILED: continuing...; fi; done
script_offset_f=fc0000
script_offset_nor=0xE2FC0000
script_size_f=0x40000
scriptaddr=3000000
soc=zynq
stderr=serial@e0000000
stdin=serial@e0000000
stdout=serial@e0000000
ubifs_boot=env exists bootubipart || env set bootubipart UBI; env exists bootubivol || env set bootubivol boot; if ubi part ${boo
tubipart} && ubifsmount ubi${devnum}:${bootubivol}; then devtype=ubi; run scan_dev_for_boot; fi
usb_boot=usb start; if usb dev ${devnum}; then devtype=usb; run scan_dev_for_boot_part; fi
vendor=xilinx

Environment size: 5849/131067 bytes
Zynq>
```

Before changing bootargs environment variable, htop as a reference with the shared usage of both CPUs

COM5 - PuTTY

```
    0[###############****        58.0%]   Tasks: 27, 8 thr; 2 running
    1[##########****             42.9%]   Load average: 3.54 1.19 0.42
  Mem[|||||||##***********98.9M/494M]   Uptime: 00:01:29
  Swp[|                     256K/512M]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+  Command
  879 root       20   0 31828  5084  2972 R 10.6  1.0  0:00.16 (haveged)
    1 root       20   0 31824  7804  5696 S  7.9  1.5  0:07.37 /sbin/init earlyp
  110 root       19  -1 51928 13072 12308 S  1.3  2.6  0:01.44 /lib/systemd/syst
  651 xilinx     20   0  7844  2688  2160 R  1.3  0.5  0:00.89 htop
  262 messagebu  20   0  6684  3020  2520 S  0.7  0.6  0:00.60 @dbus-daemon --sy
  295 syslog     20   0 25768  2940  2
  159 root       20   0 18680  3724  3012 S  0.0  0.7  0:00.67 /lib/systemd/syst
  173 systemd-r  20   0 18872  8704  5628 S  0.0  1.7  0:00.83 /lib/systemd/syst
  174 systemd-t  20   0 19960  2732  2232 S  0.0  0.5  0:00.67 /lib/systemd/syst
  208 systemd-t  20   0 19960  2732  2232 S  0.0  0.5  0:00.05 /lib/systemd/syst
  237 root       20   0 32520  4440  3516 S  0.0  0.9  0:00.05 /sbin/dhclient -1
  244 root       20   0 32520  4440  3516 S  0.0  0.9  0:00.01 /sbin/dhclient -1
  245 root       20   0 32520  4440  3516 S  0.0  0.9  0:00.00 /sbin/dhclient -1
  246 root       20   0 32520  4440  3516 S  0.0  0.9  0:00.00 /sbin/dhclient -1
  261 root       20   0  5184   924   820 S  0.0  0.2  0:00.02 /usr/sbin/cron -f

F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Change Bootargs

- *If you need to return to the default bootargs, you can find them below*
  - *https://github.com/Xilinx/PYNQ/blob/master/sdbuild/boot/meta-pynq/recipes-bsp/device-tree/files/pynq_bootargs.dtsi*
  - *bootargs = 'root=/dev/mmcblk0p2 rw earlyprintk rootfstype=ext4 rootwait devtmpfs.mount=1 uio_pdrv_genirq.of_id="generic-uio" clk_ignore_unused'*
- To edit bootargs:
  - Interrupt the boot ***{during initiasl boot, just press any key to enter into bootloader}***
  - Edit boot arguments:
    - *$ editenv bootargs* ***{before making changes to bootargs variable we can see the actual value by $ echo $bootargs}***
    - Insert arguments included the quotations all in one line:
- Bootargs (default and more) are at here

```
bootargs = 'console=ttyPS0,115200 root=/dev/mmcblk0p2 rw earlyprintk rootfstype=ext4 rootwait devtmpfs.mount=1 uio_pdrv_genirq.of_id="generic-uio" clk_ignore_unused isolcpus=1 && bootz 0x03000000 - 0x02A00000'
```

- *$ boot*
- Change bootargs to the following
  - *bootargs = 'console=ttyPS0,115200 root=/dev/mmcblk0p2 rw earlyprintk rootfstype=ext4 rootwait devtmpfs.mount=1 uio_pdrv_genirq.of_id="generic-uio" clk_ignore_unused* ***isolcpus=1*** *&& bootz 0x03000000 - 0x02A00000'*

```
Zynq> edit bootargs
edit: bootargs = 'console=ttyPS0,115200 root=/dev/mmcblk0p2 rw earlyprintk rootfstype=ext4 rootwai
t devtmpfs.mount=1 uio_pdrv_genirq.of_id="generic-uio" clk_ignore_unused isolcpus=1 && bootz 0x030
00000 - 0x02A00000'
Zynq> echo $bootargs
bootargs = console=ttyPS0,115200 root=/dev/mmcblk0p2 rw earlyprintk rootfstype=ext4 rootwait devtm
pfs.mount=1 uio_pdrv_genirq.of_id="generic-uio" clk_ignore_unused isolcpus=1 && bootz 0x03000000 -
 0x02A00000
```

- **What does isolcpus=1 do?**

Remove the specified CPUs, as defined by the cpu_number values. These are set of CPUs that the kernel process scheduler will not interact.
See htop displays No utilization of CPU1, while everything executes on CPU0:

COM5 - PuTTY

```
  0[#########################100.0%]   Tasks: 28, 10 thr; 2 running
  1[                           0.0%]   Load average: 0.42 0.25 0.32
 Mem[||||||||##***********124M/494M]   Uptime: 00:14:47
 Swp[                       0K/512M]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+  Command
 2040 root       20   0 33940 23808  9752 R 83.8  4.7  0:02.46 /usr/local/share/
    1 root       20   0 30800  7800  5696 S  5.8  1.5  0:12.76 /sbin/init bootar
  575 root       20   0  125M 67160 16864 S  0.6 13.3  0:28.95 /usr/local/share/
 1953 xilinx     20   0  7736  2428  2060 R  0.6  0.5  0:00.78 htop
 2042 root       20   0  125M 67160 16864 S  0.6 13.3  0:00.02 /usr/local/share/
  108 root       19  -1 60208 13432 12636 S  0.0  2.7  0:02.15 /lib/systemd/syst
  147 root       20   0 18552  3672  3020 S  0.0  0.7  0:00.59 /lib/systemd/syst
  171 systemd-r  20   0 18872  8708  5628 S  0.0  1.7  0:01.58 /lib/systemd/syst
  173 systemd-t  20   0 19960  2732  2232 S  0.0  0.5  0:01.17 /lib/systemd/syst
```

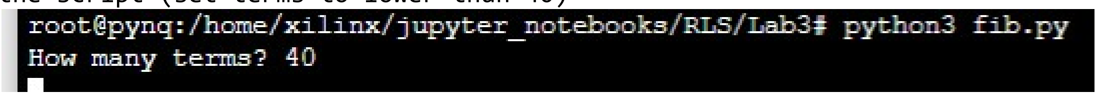
- **What would isolcpus=0 do?**

The kernel process scheduler will interact with all CPU's (in this case CPU0 & CPU1)-shared load

Heavy CPU Utilization

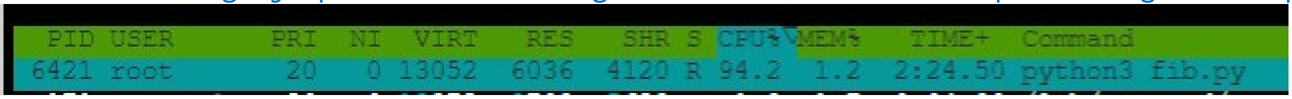
● Download *fib.py* from here. This is a recursive implementation for generating Fibonacci sequences. We just do not print the results.
● Make sure your board is booted with custom bootargs above including *isolcpus=1*
1) Open two terminals (Jupyter):
● Terminal 1: run *htop* to monitor CPU utilization

● Terminal 2: run *$ python3 fib.py* and monitor CPU utilization and time spent for running the script (set terms to lower than 40)



● **Describe the results of *htop*.**

The CPU0's usage jumped to a 100% usage because the intensive processing of fib.py



{noticed the CPU1 remains 0% (no use at all)}

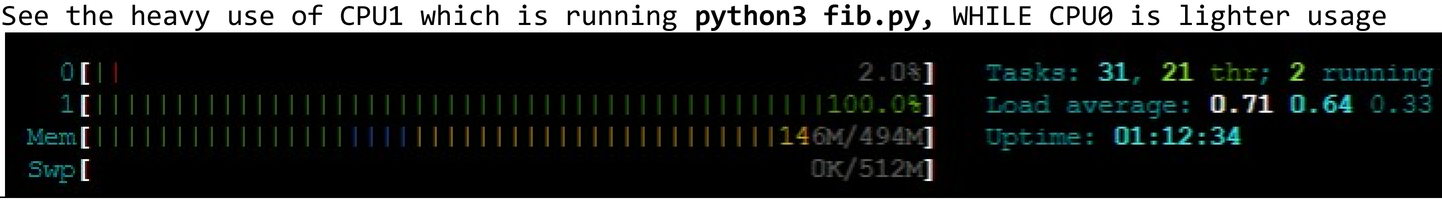
2) Repeat the previous part, but this time use *taskset* to use CPU1:
● Terminal 2: run *$ taskset -c 1 python3 fib.py* and monitor CPU utilization and time spent for running the script

● **Describe the results of *htop*. Specifically, what's different from running it in 1)?**

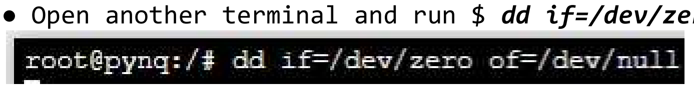
Since this time we attached the process "python3 fib.py" to CPU1 by issuing the command:

$ taskset -c 1 python3 fib.py

See the heavy use of CPU1 which is running **python3 fib.py,** WHILE CPU0 is lighter usage



3) Heavy Utilization on CPU0:
● Open another terminal and run $ ***dd if=/dev/zero of=/dev/null***



● Repeat parts 1 and 2
● **Describe the results of *htop*.**

After issuing command *"$ dd if=/dev/zero of=/dev/null", now the usage load is shared between* ***CPU0 & CPU1***

ARM Performance Monitoring (C++)

- Download kernel_modules folder
- Read through CPUcntr.c and reference the ARM documentation for the PMU registers here to answer the following question.

○ **According to the ARM docs, what does the following line do? Are they written in assembly code, python, C, or C++?**

■ *asm(“MCR p15, 0, 1, c9, c14, 0\n\t”);*

This code is Assembly language;
asm(“MCR p15, 0, 1, c9, c14, 0\n\t”); {program the performance-counter control-register}

The PMU module enables users to access performance counter, Providing Access to PMU for Applications on both CPUs, this kernel object can enable user-mode for PMU.
ARM Performance Monitoring Unit (PMU)
- PMU is an event counting hardware which can be used to profile and benchmark code
- Cortex-A9 PMU provides six counters
- Each counter can count any of the 58 events available in the Cortex-A9 processor
- These counters can be accessed through debugging tools or directly through the CP15 Registers

Accessing ARM PMU
- By default user space (applications), do not have access to PMU
- Kernel has access to PMU

MCR: Move to Coprocessor from ARM Register, PMU coprocID is p15

System control processor registers overview

This section gives details of all the registers in the system control coprocessor. The section presents a summary of the registers and detailed descriptions in register order of CRn, Opcode_1, CRm, Opcode_2.

You can access CP15 registers with MRC and MCR instructions, as described in *Use of the system control coprocessor*:

```
MCR{cond} p15,<Opcode_1>,<Rd>,<CRn>,<CRm>,<Opcode_2>
```

```
MRC{cond} p15,<Opcode_1>,<Rd>,<CRn>,<CRm>,<Opcode_2>
```

You can access the ARM PMU (ARM Performance Monitor Unit) counters, and their associated control registers through the internal CP15 interface , and through the APB ,using the relevant offset when PADDRDBG[12]=1

○ Compile and insert the kernel module following the instructions from the README file.
- Download clock_example folder
- Read through *include/cycletime.h* and take note of the functions to initialize the counters and get the cyclecount (what datatype do they return, what parameters do they take)

○ **What does the following line do?**

■ *asm volatile ("MRC p15, 0, %0, c9, c13, 0\n\t" : "=r"(value));*

This line will return the cyclecount

{{{ Per Teacher Assistant, we stop here because compiler change issues with the PYNQ board}}}

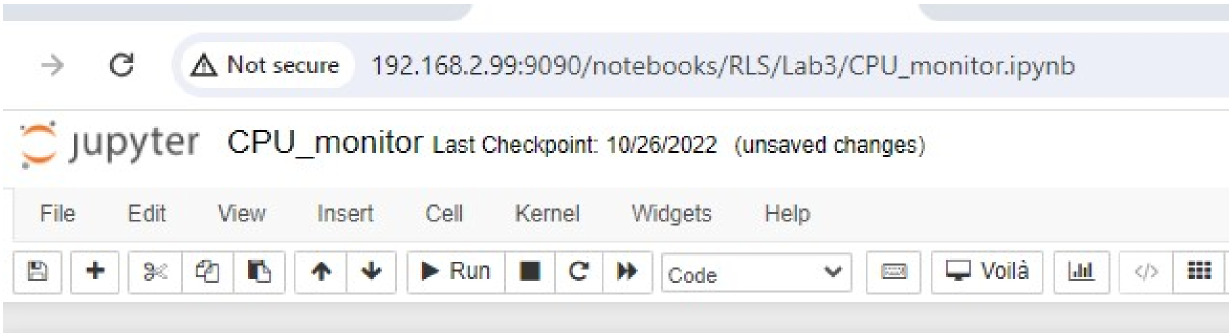
Complete the code in *src/main.cpp*. These instructions are for those who have never coded in C++

○ **Declare 2 variables (cpu_before, cpu_after) of the correct datatype**
○ **Initialize the counter**
○ **Get the cyclecount ‘before’ sleeping**
○ **Get the cyclecount ‘after’ sleeping**
○ **Print the difference number of counts between starting and stopping the counter**
- After completing the code, open a jupyter terminal and change directory to *clock_examples/*
- Run *$ make* to compile the code
- Run the code with *$ ./lab4 <delay-time-seconds>*
- **Change the delay time and note down the different cpu cycles as well as the different timers.**

{{{ Per Teacher Assistant, we stop here because compiler change issues with the PYNQ board}}}

Jupyter Notebook CPU Monitoring (OPTIONAL)

- Download CPU_monitor.ipynb from here. This is an interactive implementation for plotting in a loop. Running this notebook is a computationally heavy task for your CPU, therefore you do not need to run any additional process to utilize your CPU0.



In [2]:
```
import time
import pylab as pl
from IPython import display
import psutil
import matplotlib.pyplot as plt
import numpy as np
```

In [3]:
```
psutil.cpu_percent(percpu=True)
```

Out[3]: [100.0, 0.0]

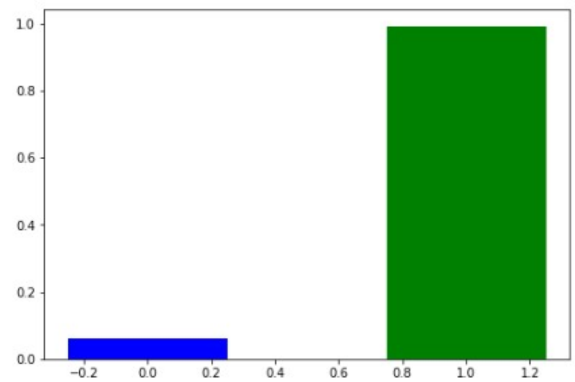
In [4]:
```
%matplotlib inline

X = np.arange(1)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
for i in range(10):
    data = psutil.cpu_percent(percpu=True)
    ax.cla()
    ax.bar(X + 0.0, data[0]/100, color = 'b', width = 0.5)
    ax.bar(X + 1.0, data[1]/100, color = 'g', width = 0.5)
    display.clear_output(wait=True)
    display.display(plt.gcf())
plt.clf()
```

- Create a Jupyter notebook
  - Use the *os* library to create a python program that accepts a number from user input (0 or 1) and runs *fib.py* on a specific core (0 or 1).
  - *Hint: look at the os.system() call and remember the 'taskset' function we've used previously.*

```
import os
cpu = input('enter CPU number (0/1)?')
str = "taskset -c "+cpu+" python3 fib.py"
print(str)
os.system(str)
```

- You should have two notebooks running: 1) CPU_monitor, 2) CPU_select
  - **Compare your observations between using Jupyter notebook CPU_monitor and linux command *htop* for monitoring CPU utilization.**

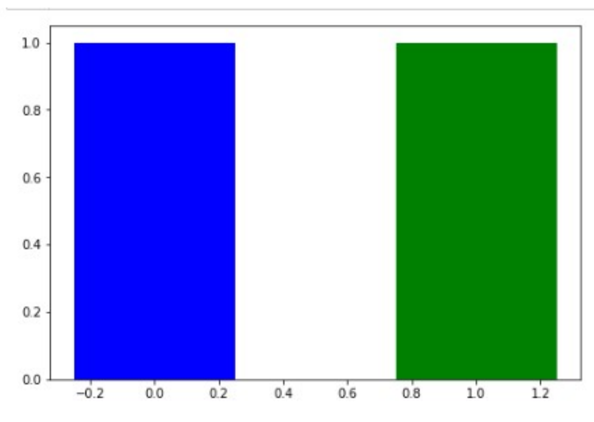
Before the test:



CPU1 running python3 fib.py:

```
[*]:
import os
cpu = input('enter CPU number (0/1)?')
str = "taskset -c "+cpu+" python3 fib.py"
print(str)
os.system(str)
```

```
enter CPU number (0/1)?1
taskset -c 1 python3 fib.py
```



HTOP application on console responses significantly faster that the monitoring application on Jupyter botebook, possible reason are that htop app maybe a c++ compiled application, while the python jupyter code is translated code into c++ libraries on the fly, all these plus the graphical overhead takes a lot of CPU/ Clock resources